Title: Distributed_Applications (28.05.2013)
Date: Tue May 28 14:31:52 CEST 2013
Duration: 88:17 min
Pages: 26

Servlets

Servlet Properties
Servlet Lifecycle
HttpServletRequest
Structure of a Servlet

```
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {
  // called by the servlet engine to initialize servlet
  public void init() throws ServletException {
  }
  // process the HTTP GET request
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  }
  // process the HTTP POST request
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  }
  // called by the servlet engine to release the resource
  public void destroy() {
  }
  // other methods

  // Example - CurrentTime
  // Example - Registration of Students
```

HttpServlet Interface

HttpServletRequest inherits abstract class GenericServlet which implements interfaces Servlet and ServletConfig.

GenericServlet defines a generic protocol-independent servlet

HttpServletRequest defines a servlet for the HTTP protocol

- doGet (req, httpServletResponse): void
- doPost (req, httpServletResponse): void
- doDelete (req, httpServletResponse): void

- init (config, servletConfig): void
- service (req, HttpServletRequest, HttpServletResponse): void
- destroy(): void

- getParameter (name: String): String
- getParametersNames(): Enumeration
- getServletContext(): ServletContext
- getServletName(): String

doGet is invoked to respond to a GET request

doPost is invoked to respond to a POST request

doDelete is invoked to respond to a DELETE request; normally used to delete a file on the server
Servlet

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class GetParameters extends HttpServlet {
  /** process the HTTP GET request */
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // obtain parameters from the client
    String lastName = request.getParameter("lastName");
    String firstName = request.getParameter("firstName");
    String gender = request.getParameter("gender");
    String major = request.getParameter("major");
    String[] minors = request.getParameterValues("minor");
    String tennis = request.getParameter("tennis");
    String soccer = request.getParameter("soccer");
    String golf = request.getParameter("golf");
    String remarks = request.getParameter("remarks");
    out.println("Last Name: " + lastName + " First Name: " + firstName + " Gender: " + gender + " Major: " + major + " Minor(s): " + minors.length + " Tennis: " + tennis + " Soccer: " + soccer + " Golf: " + golf + " Remarks: " + remarks);
  }
}
```

Servlets

Servlets (Java Servlets) are programs invoked and executed on the server host. They are used to extend the functionality of the server.

- **Servlet Properties**
- **Servlet Lifecycle**
- **HttpServletRequest Interface**
- **Structure of a Servlet**

Basic mechanisms for distributed applications

**Issues**

The following section discusses several important basic issues of distributed applications.

- Data representation in heterogeneous environments
- Discussion of an execution model for distributed applications
- What is the appropriate error handling?
- What are the characteristics of distributed transactions?
- What are the basic aspects of group communication (e.g., algorithms used by ISIS)?
  - How are messages propagated and delivered within a process group in order to maintain a consistent state?

**External data representation**

- **Time**
- **Distributed execution model**
- **Failure handling in distributed applications**
- **Distributed transactions**
- **Group communication**
- **Distributed Consensus**
- **Authentication service Kerberos**
Heterogeneous environment means different data representations

Requirement to enable data transformation.

Marshalling and unmarshalling

Centralized transformation

Decentralized transformation

Common external data representation

XML as common data representation

Marshalling and unmarshalling

Centralized transformation

Decentralized transformation

Common external data representation

XML as common data representation
For the representation of numbers in main memory, one of the following methods are generally used.

"Little endian" representation: the lower part of a number is stored in the lower memory area.

"Big endian" representation: the higher part of a number is stored in the lower memory area, e.g., the Sun-SPARC architecture.

Example representation of the number 1347

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>1000</th>
<th>1001</th>
<th>1002</th>
<th>1003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Endian</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>0100000000000000</td>
</tr>
<tr>
<td>Little Endian</td>
<td>0100000000000000</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

Convention for network transfer, numbers which encompass several bytes are structured according to a well-defined representation, such as "big endian".

Two aspects of a common external data representation are of importance:

- A machine-independent format for data representation, and
- A language for description of complex data structures.

Examples: xDR ("External Data Representation") by Sun and ASN.1 (Abstract Syntax Notation). Other formats are

Java's common data representation: structured and primitive types can be passed as arguments and results.

Java's object serialization: flattening of single objects or tree of objects.

**Representation of numbers**

**External representation of strings**

**External representation of arrays**

**Transfer of pointers**

There are different internal representations for strings:

- **C ("abc")**
  - C: 0x20 0x61 0x63 0x68
  - ASCII: a

- **Pascal ("abc")**
  - ASCII: a

**Standard external representation:**

- Length: n
- Byte: 0
- Byte: 1
- Byte: n-1
- 0
- 0

\[
\text{length} = \text{byte}_0 + \text{byte}_1 + \ldots + \text{byte}_{n-1} + 0 + 0
\]

Request for the invocation of the Java method
echoString("cat")

SOAP body of request that sends a string.

```xml
<soap:Body>
  <ns:echoString>
    <a xmlns="http://tempuri.org/mapping.server.Primitive">cat</a>
  </ns:echoString>
</soap:Body>
```
Complex data types can be mapped to XML for transmission across the network.

**Example: primitive datatypes**
- boolean, byte, unsignedShort (used for char), int, long, float, string, ...

**Example: array datatype**
- SOAP provides built-in support for encoding arrays.
- SOAP platforms provide API for creating custom mapping.
  - e.g. `writeSchema` to specify an XML schema definition

```
<soap:Body>
  <m:echoInts xmlns:m='http://tempuri.org/mapping.server.Array'>
    <m:ints>
      <m:int id='id0'>
        <soapenc:root='6' xsi:type='soapenc:Array'>
        </soapenc:root>
        <m:int id='id1'>
          <xsd:int>1</xsd:int>
        </m:int>
        <m:int id='id2'>
          <xsd:int>2</xsd:int>
        </m:int>
        <m:int id='id3'>
          <xsd:int>3</xsd:int>
        </m:int>
      </m:ints>
    </m:ints>
  </m:echoInts>
</soap:Body>
```

**External data representation**

Heterogeneous environment means different data representations

- Requirement to enable data transformation.
- Independence from hardware characteristics while exchanging messages: use of external data representation.
- Marshalling and unmarshalling
- Centralized transformation
- Decentralized transformation
- Common external data representation
- XML as common data representation
Time

Time is an important and interesting issue in distributed systems.

We need to measure time accurately:
- to know the time an event occurred at a computer
- to do this we need to synchronize its clock with an authoritative external clock

Algorithms for clock synchronization useful for
- concurrency control based on timestamp ordering
- authenticity of requests e.g. in Kerberos

Three notions of time:
- time seen by an external observer → global clock of perfect accuracy.
- time seen on clocks of individual processes,
- logical notion of time: event a occurs before event b.

Introduction
Synchronizing physical clocks

Timestamp

To timestamp events, we use the computer’s clock.
1. At real time t, the operating system reads the time on the computer’s hardware clock H(t).
2. It calculates the time on its software clock C(t) = H(t) + b
   e.g. a 64 bit number giving nanoseconds since some “base time”
   in general, the clock is not completely accurate,
   but if C behaves well enough, it can be used to timestamp events at p.

Introduction

Each computer in a distributed system (DS) has its own internal clock
used by local processes to obtain the value of the current time
processes on different computers can timestamp their events
but clocks on different computers may give different times
computer clocks drift from perfect time and their drift rates differ from one another.

clock drift rate: the relative amount that a computer clock differs from a perfect clock

Even if clocks on all computers in a DS are set to the same time, their clocks will eventually vary quite
significantly unless corrections are applied.

Timestamp

Skew between clocks

Coordinated Universal Time (UTC)

Introduction

Each computer in a distributed system (DS) has its own internal clock
used by local processes to obtain the value of the current time
processes on different computers can timestamp their events
but clocks on different computers may give different times
computer clocks drift from perfect time and their drift rates differ from one another.

clock drift rate: the relative amount that a computer clock differs from a perfect clock

Even if clocks on all computers in a DS are set to the same time, their clocks will eventually vary quite
significantly unless corrections are applied.

Timestamp

Skew between clocks

Coordinated Universal Time (UTC)
Time is an important and interesting issue in distributed systems.

We need to measure time accurately:
- to know the time an event occurred at a computer
- to do this we need to synchronize its clock with an authoritative external clock.

Algorithms for clock synchronization useful for concurrency control based on timestamp ordering authenticity of requests e.g. in Kerberos.

Three notions of time:
- time seen by an external observer → global clock of perfect accuracy.
- However, there is no global clock in a distributed system
- time seen on clocks of individual processes.
- logical notion of time: event a occurs before event b.

Introduction

Synchronizing physical clocks

External synchronization

A computer's clock Ci is synchronized with an external authoritative time source Di so that:
\[ |Si(t) - Di(t_i)| < D \] for \( i = 1, 2, ..., N \) over an interval of real time t.

The clocks Ci are accurate to within the bound D.

Internal synchronization

The clocks of a pair of computers are synchronized with one another so that:
\[ |Ci(t) - Cj(t)| < D \] for \( i, j = 1, 2, ..., N \) over an interval of real time t.

The clocks Ci and Cj agree within the bound D.

Internally synchronized clocks are not necessarily externally synchronized, as they may drift collectively.

If the set of processes P is synchronized externally within a bound D, it is also internally synchronized within bound 2D.

Network Time Protocol (NTP)

Precision Time Protocol (PTP)